USING TRANSACTED WRITES AND CACHING MECHANISM TO IMPROVE WRITE PERFORMANCE IN MULTI-LEVEL CELL FLASH MEMORY

BACKGROUND

[0001] Non-volatile memory products for electronic equipments, such as cell phones, digital cameras, computers, etc., are widely used today. When writing a file to the non-volatile memory, data fragments, sequence tables and their associated headers may be written to the non-volatile memory one by one. Here, the data fragments store user data of the file and the sequence tables may comprises sequence table entries to store memory locations of the data fragments. A sequence table entry may comprise a data field for a memory location, and, an entry allocating field and entry valid field to protect the sequence table entry in case of power loss.

BRIEF DESCRIPTION OF THE DRAWINGS

[0002] The invention described herein is illustrated by way of example and not by way of limitation in the accompanying figures. For simplicity and clarity of illustration, elements illustrated in the figures are not necessarily drawn to scale. For example, the dimensions of some elements may be exaggerated relative to other elements for clarity. Further, where considered appropriate, reference labels have been repeated among the figures to indicate corresponding or analogous elements.

[0003] Fig.1 illustrates a memory writing device;

[0004] Fig. 2 illustrates an embodiment of a file system maintained in a non-volatile memory of the memory writing device of Fig. 1;

[0005] Fig. 3 illustrates an embodiment of a sequence table temporarily stored in a volatile memory of the memory writing device of Fig. 1;

[0006] Fig. 4a illustrates a transaction indicator in the file system of Fig. 2;

[0007] Fig. 4b illustrates a sequence table entry in the file system of Fig. 2;

[0008] Fig. 4c illustrates a sequence table header in the file system of Fig. 2;

[0009] Fig. 4d illustrate a data fragment header in the file system of Fig. 2;

[0010] Fig.5 illustrates an embodiment of a memory writing method that may be used by the device of Fig. 1 ;

[0011] Fig. 6 illustrates a comparison between two memory writing methods.

DETAILED DESCRIPTION

[0012] The following description describes techniques for memory writing method and system. In the following description, numerous specific details such as logic implementations, pseudo-code, means to specify operands, resource partitioning/sharing/duplication implementations, types and interrelationships of system components, and logic partitioning/integration choices are set forth in order to provide a more thorough understanding of the present invention. However, the invention may be practiced without such specific details. In other instances, control structures, gate level circuits and full software instruction sequences have not been shown in detail in order not to obscure the invention. Those of ordinary skill in the art, with the included descriptions, will be able to implement appropriate functionality without undue experimentation.

[0013] References in the specification to "one embodiment", "an embodiment", "an example embodiment", etc., indicate that the embodiment described may

include a particular feature, structure, or characteristic, but every embodiment may not necessarily include the particular feature, structure, or characteristic. Moreover, such phrases are not necessarily referring to the same embodiment. Further, when a particular feature, structure, or characteristic is described in connection with an embodiment, it is submitted that it is within the knowledge of one skilled in the art to effect such feature, structure, or characteristic in connection with other embodiments whether or not explicitly described.

[0014] Embodiments of the invention may be implemented in hardware, firmware, software, or any combination thereof. Embodiments of the invention may also be implemented as instructions stored on a machine-readable medium, which may be read and executed by one or more processors. A machine-readable medium may include any mechanism for storing or transmitting information in a form readable by a machine (e.g., a computing device). For example, a machine-readable medium may include read only memory (ROM); random access memory (RAM); magnetic disk storage media; optical storage media; flash memory devices; and others.

[0015] Fig. 1 shows a memory writing device 100 according to one embodiment . The device 100 includes a processor 101, a non-volatile memory 102, and a volatile memory 103. The processor 101 may be any type of processor adapted to perform operations to the non-volatile memory 102. For example, the processor 101 may be a microprocessor, a digital signal processor, a microcontroller, or the like. The non-volatile memory 102 may comprises a flash memory, such as NOR flash memory, NAND flash memory. A volatile memory 103 may comprises various types of random access memories (RAMs), for

example, dynamic random access memory (DRAM), synchronous direct random access memory (SDRAM), double data rate (DDR) SDRAMs, or other memories.

[0016] The processor 101, the non-volatile memory 102 and the volatile memory103 may be coupled by buses 104-105. In some embodiments, the processor 101, the non-volatile memory 102 and the volatile memory 103 may be included on an integrated circuit board, and the buses 104-105 may be implemented using traces on the circuit board.

[0017] According to various embodiments, the processor 101 may perform operations to non-volatile memory 102 and volatile memory 103. In one embodiment, the processor 101 may control writing a partial file or a whole file to the non-volatile memory 102. The file may comprise a plurality of data fragments, at least one sequence table including a plurality of sequence table entries to identify memory locations of the data fragments, and other associated information. Before being written to the non-volatile memory 102, the sequence table entries may be temporarily stored to the volatile memory 103 and then may be transferred from the volatile memory 103 to the non-volatile memory 102 under the control of the processor 101.

[0018] Referring now to Figs. 2-3 and Figs. 4a-4d, a file system of the non-volatile memory 102 and a sequence table temporarily stored in the volatile memory 103 will be described in detail. Figs. 2 and 3 respectively show the file system stored in the non-volatile memory 102 and the sequence table temporarily stored in the volatile memory 103. Figs. 4a-4d respectively show a transaction indicator (TI), sequence table entry (STE), sequence table header (STH) and data fragment header (DFH) of the file system shown in Fig. 2.

[0019] According to various embodiments, the file system 200 maintained in the non-volatile memory may hold files written to the non-volatile memory and any associated information about the files. The file system 200 may include a file information structure 210 and a number of storage blocks 220, 250, 260 and 280, etc. For NOR flash memory manufactured by Intel Corporate, Santa Clara, California, a storage block may comprise 64k or 128k bytes. For NAND flash memory manufactured by SAMSUNG, Korea, a block may comprise 16k or 32k bytes. As shown in Fig. 2, the file information structure 210 may include a transaction indicator (TI) 215 to indicate states of a transaction for writing a file to the non-volatile memory 102, including a begin state and end state of the transaction. The file information structure 210 may further include a number of sequence table pointers (STP) (e.g., STPs 211, 212 and 213) to respectively point to sequence tables (ST) in the storage blocks of the non-volatile memory 102 (e.g., STs 221, 252 and 253 in the storage blocks 220 and 250). The file information structure 210 may also include the filename, creation date, size of a file, or any other information relating to a file. Storage blocks 220, 250, 260 and 280 may represent blocks of memory of the non-volatile memory 102. In some embodiments, each storage block may be divided into a number of fragments to hold a portion of a file. For example, as shown in Fig. 2, storage blocks 220 and 250 may include fragments to hold the sequence tables (e.g., STs 221, 252 and 253) of the file, and storage blocks 260 and 280 may include fragments to hold user data (e.g., data fragments 261, 262 and 283) of the file. For NOR flash memory manufactured by Intel Corporate, Santa Clara, California, US, a

fragment may comprise 512 or 1k bytes. It shall be understood that storage blocks may be used to hold other data than those of the file.

[0020] Moreover, each sequence table may have a sequence table header (STH) associated therewith, and each data fragment may have a data fragment header (DFH) associated therewith. For example, as shown in Fig. 2, the sequence tables (ST) 221, 252 and 253 have their individually associated headers, i.e., the sequence table headers (STH) 221', 252' and 253'. Similarly, the data fragments (DF) 261, 262 and 283 have their individually associated headers, i.e., the data fragment headers (DFH) 261', 262' and 283'. Detailed structure of the sequence table (ST) 221 is shown in Fig. 2. For simplicity, detailed structures of the sequence tables (ST) 252 and 253 which are the same as that of the sequence table (ST) 221 are omitted in Fig. 2. As depicted, the sequence table 221 may include a number of sequence table entries (STE), for example, sequence table entries (STE) 2211, 2212 and 2213, to store memory locations of corresponding data fragments (DF) 261, 262 and 283. In one embodiment, a sequence table may be the same size as a fragment which may be less than the size of a storage block. Accordingly, a single sequence table may suffice for a file with a small size; however, a file with a relatively large size may include more than one sequence table, such as sequence tables (ST) 221, 252 and 253.

[0021] It shall be appreciated that a file in the file system 200 may include any number of storage blocks to hold any number of data fragments and sequence tables. A sequence table may include any number of sequence table entries to point to the data fragments, as long as within its size limitation. The file may

further include any other kinds of data structures than those shown in Fig. 2. In another embodiment, the file information structure 210 of the file system 200 may include a pointer to point to a storage block in the file system 200 which may be used to hold sequence table pointers to point to the sequence tables. The file information structure 210 may further include different transaction indicators for different files in the file system 200. Other embodiments may implement other modifications and variations to the structure of the file system 200.

[0022] Detailed structure of the transaction indicator (TI) 215 is shown in Fig. 4a. As depicted, the transaction indicator 215 may include two transaction state fields: a transaction begin field, indicating the begin state of the transaction for writing a file; and a transaction end field, indicating the end state of the transaction. In an embodiment, the transaction begin field may comprises 2 bits to represent the transaction begin state, and the transaction end field may comprises 2 bits to represent the transaction end state.

[0023] In an embodiment, the transaction begin field and transaction end field of the transaction indicator (TI) 215 may each include 1 bit. In another embodiment, the transaction begin field may have 2 bits while the transaction end field may have 1 bit. Other embodiments may utilize transaction indicators (TI) 215 having a different structure and/or a different number of bits.

[0024] The transaction indicator may support a power loss recovery feature to keep data consistency of the file to be written in the non-volatile memory 102. Particularly, when power loss happens after the transaction for writing the file is started, but before the transaction ends, parts of the file which have been written

to the non-volatile memory may be deleted automatically by the power loss recovery feature.

[0025] Referring now to Fig. 4b, the detailed structure of the sequence table entry 221 is shown. In Fig. 4b, the sequence table entry may include a data entry field to identify a memory location of a data fragment. Before a sequence table comprising a number of the sequence table entries is written to the non-volatile memory, the sequence table may be temporarily stored in the volatile memory 103. Fig. 3 shows the data structure of a sequence table temporarily stored in the volatile memory 103. It can be seen that the sequence table stored in volatile memory 103 has the same structure as that in the non-volatile memory 102.

[0026] Each sequence table has a sequence table header (STH) associated therewith. Detailed data structure of the sequence table header 221' is shown in Fig. 4c. In Fig. 4c, the sequence table header has two fields representing two sequence table header states: a sequence table header allocating field (st_hdr_allocating field) to represent 'sequence table header allocating' state; a sequence table header valid field (st_hdr_valid field) to represent 'sequence table header valid' state. Data structures of the sequence table headers 252' and 253' are the same as that of the sequence table header 221'.

[0027] Each data fragment may have a data fragment header (DFH) associated therewith. Detailed data structure of the data fragment header 261' is shown in Fig. 4d. As depicted, the data fragment header may include two fields representing two data fragment header states: a data fragment header allocating field (df_hdr_allocating field) to represent 'data fragment header allocating' state; a data fragment header valid field (df_hdr_valid field) to represent 'data fragment

header valid' state. The data structures of the data fragment headers (DFH) 262' and 283' may be implemented in a manner similar to the data fragment header (DFH) 261'.

[0028] It shall be appreciated that the sequence table header and/or data fragment header may further include other data structures, such as a header invalid field (hdr_invalid field) to represent 'header invalid' state. The sequence table header and data fragment header may support a power loss recovery feature for their associated sequence table and data fragment in the non-volatile memory 102. Namely, if power loss happens before validation of a header, its associated sequence table or data fragment may be deleted automatically.

[0029] Now referring to Fig. 5, a method of writing a file to the non-volatile memory according to an embodiment will be described in detail below, the file having the data structure as shown in Fig. 2. The method may be implemented by the device 100 as shown in Fig. 1.

[0030] In block 501, the processor 101 may start a transaction for writing a file by changing state of the transaction indicator (TI) 215 in the non-volatile memory 102 to a transaction begin state. For implementation, in an embodiment where the transaction indicator 215 includes a transaction begin field having 2 bits to identify the transaction begin state, the processor 101 may write to the transaction begin field, thereby changing the 2 state bits in the transaction begin field from "11" to "00" to indicate begin of the transaction for writing the file. This change from "11" to "00" may be accomplished using bit twiddling write mode which is a special word programming write mode supported by some non-volatile memory devices. Namely, only two bits of a word are programmed in the bit

twiddling mode. However, other embodiments may utilize a different data structure of the transaction indicator and/or a different write mode.

[0031] In block 502, the processor 101 may control changing state of a data fragment header (e.g., DFH 261' in Fig. 2) to a 'data fragment header allocating' state which indicates a fragment of a block in the non-volatile memory 102 is allocated to hold a data fragment of the file. Again, this change in state may be accomplished by writing to a data fragment header allocating field (df_hdr_allocating field) of the data fragment header in bit twiddling write mode.

[0032] In block 503, the processor 101 may control writing to the non-volatile memory 102 a data fragment associated with the data fragment header in block 502. For example, as shown in Fig. 2, data fragment (DF) 261 associated with data fragment header (DFH) 261' in block 502 may be written to the non-volatile memory in block 503. This writing of the data fragment may be accomplished in one embodiment with buffer programming write mode to write user data of the file.

[0033] In block 504, the processor 101 may control writing a sequence table entry to the volatile memory 103. The sequence table entry may comprise a data entry field to identify the memory location of the corresponding data fragment written in block 503. For example, the sequence table entry (STE) 2211 as shown in Fig. 3, which identifies the memory location of the corresponding data fragment (DF) 261 written in block 503, may be written to the volatile memory 103 in block 504. Since the sequence table entry may be written to a volatile memory, the sequence table entry may not need to add power loss recovery information for each sequence table entry. That is to say, the sequence table entry may not

comprise fields to identify entry states, such as entry allocating state and entry valid state, for the purpose of power loss recovery.

[0034] In block 505, the processor 101 may determine whether all of the data fragments which carry user data of the file are written to the non-volatile memory. Namely, the processor 101 may determine whether writing the user data of the file to the non-volatile memory is completed. If not, the processor 101 may continue to block 506 where the processor 101 may determine whether the sequence table temporarily stored in the volatile memory 103 (e.g., sequence table 221 in Fig. 3) is full. Since a sequence table may have a limited size, a file with a large size may need more than one sequence table to accommodate a large amount of sequence table entries to identify the memory locations for all of the data fragments of the file. If the processor 101 determines that the sequence table temporarily stored in the volatile memory 103 in not full, the processor 101 may return to block 502 to allocate another data fragment. Namely, the processor 101 may continue to write data fragments to the non-volatile memory 102 and to write their corresponding sequence table entries to the volatile memory 103.

[0035] If the sequence table stored in the volatile memory 103 is determined to be full in block 506, the processor 101 may continue to block 507 to allocate another sequence table header. Namely, the processor 101 in block 507 may change state of a sequence table header to 'sequence table header allocating' state in the non-volatile memory 102, indicating that a fragment of a block in the non-volatile memory is allocated to hold a sequence table which may be written to the non-volatile memory. For example, the sequence table 221 temporarily stored in the volatile memory 103 (STH) 221' in Fig. 2 may be changed to a

'sequence table header allocating' state. This change in state may be implemented in one embodiment by writing a sequence table header allocating field (st_hdr_allocating field) of the sequence table header (STH) 221' in bit twiddling write mode.

[0036] Then, in block 508, the processor 101 may control writing of the sequence table temporarily stored in the volatile memory 103 to the non-volatile memory 102. For example, the sequence table (ST) 221 in Fig. 3 may be written from the volatile memory to the non-volatile memory in block 508. This writing of the sequence table may be implement in one embodiment with buffer programming write mode. By completion of block 508, a part of the file corresponding to one sequence table is written to the non-volatile memory.

[0037] Then, the processor 101 may return to block 502 to continue writing other parts of the file corresponding to other sequence tables such as, for example, the sequence tables (ST) 252 and 253 as shown in Fig. 2. However, modifications and variations to the above are possible. For instance, when a size limitation for a sequence table is set to be large enough to accommodate locations of data fragments of the file, blocks 506-508 may be omitted from Fig. 5.

[0038] If, in block 505, the processor 101 determines that writing the user data of the file to the non-volatile memory is completed, the processor may continue to block 509. Namely, the processor 101 may control changing state of a sequence table header in the non-volatile memory 102 to a sequence table header allocating state, which indicates a fragment of a block in the non-volatile memory is allocated to hold a sequence table which may be written to the non-volatile memory. In an embodiment that that may utilize more than one sequence table

for the file, the fragment allocated in block 509 may be used to hold the last sequence table of the file. For example, a file may comprise three sequence tables (ST) 221, 252 and 253 as shown in Fig. 2. After parts of the file corresponding to the sequence tables 212 and 252 are written to the non-volatile memory 102 and processor 101 is in the process of writing the last part of the file corresponding to the last sequence table 253, the processor 101 may determine that all of the data fragment carrying user data of the file have been written to the non-volatile memory. The processor 101 may change the state of a sequence table header associated with the last sequence table, e.g., sequence table header (STH) 253' to sequence table header allocating state which indicates a fragment of a storage block is allocated to hold the sequence table 253 in the non-volatile memory 102. This allocating of the sequence table header may be implemented by one embodiment by writing a sequence table header allocating field of the sequence table header (STH) 253' in bit twiddling write mode.

[0039] Then, in block 510, the processor 101 may control writing of the sequence table temporarily stored in the volatile memory 103 to the non-volatile memory 102. In an embodiment, the last sequence table 253 which is temporarily stored in the volatile memory 103 may be written to the non-volatile memory 102 in block 510.

[0040] Then, in block 511, the processor 101 may control changing the state of the transaction indicator 215 to a transaction end state. For implementation, in an embodiment where the transaction indicator 215 includes a transaction end field having 2 bits to identify the transaction end state as shown in Fig. 4a, the processor 101 may write to the transaction end field, thereby changing the 2

state bits in the transaction end field from "11" to "00" to indicate end of writing the file. The following table 1 shows how states of the transaction corresponds to the pair of the transaction begin and transaction end fields according with an embodiment.

Transaction Begin Field	Transaction End Field	Transaction State
00	11	begin
00	00	end

Table 1

[0041] This ending of the transaction may be implemented by an embodiment via a bit twiddling write mode. However, other embodiments may utilize a transaction indicator with a different structure, a different number of bits, and/or a different write mode.

[0042] Then, in block 512, the processor 101 may change the state of data fragment headers associated with the data fragments of the file which have been written to the non-volatile memory from header allocating state to a header valid state. Completion of block 512 indicates that the fragments of the blocks, which have been allocated in block 502 to hold the data fragments of the file, are validated. This validation may be implemented by an embodiment by writing header valid fields of the data fragment headers associated with the data fragments, which have been written to the non-volatile memory, in bit twiddling write mode.

[0043] Then, in block 513, the processor 101 may control changing state of the sequence table header(s) associated with the sequence table(s) of the file from the sequence table header allocating state to the sequence table header valid

state. In an embodiment that the file has only one sequence table, its associated sequence table header is validated in block 513. In another embodiment that the file has more than one sequence tables, their associated sequence table headers are validated in block 513. Completion of block 513 indicates that the fragment(s) of storage block(s), which had been allocated in blocks 507 and/or 509 to hold the sequence table(s) of the file, is validated. This validation may be implemented in one embodiment by writing sequence table header valid field(s) of the sequence table header(s) associated with the sequence table(s) in bit twiddling write mode.

[0044] It can be seen from the flow chart of Fig. 5 that the data fragment headers, sequence table header(s) and the transaction indicator may support a power loss recovery feature. Namely, if the power loss occurs when the transaction of writing the file begins but not ends, which corresponding to the state of the transaction indicator is changed to the transaction begin state, the data fragments and sequence table(s) of the file, whose associated headers have not been validated yet, may be deleted from the non-volatile memory 102.

[0045] If the power loss occurs when the transaction of writing the file ends, which corresponding to the state of the transaction indicator is changed to the transaction end state, the data fragments and sequence table(s) of the file may be maintained in the non-volatile memory 102 by validating their associated headers. By this way, data consistency for the whole file may be maintained.

[0046] However, other embodiments may utilize modifications and variations of the above described power loss recovery feature. For instance, when data consistency is desired on sequence table level but not on file level, the

transaction indicator may be omitted. In such case, after each part of a file corresponding to each sequence table is written to the non-volatile memory, including writing data fragments of that part to the non-volatile memory and writing the corresponding sequence table from the volatile memory to the non-volatile memory, each of the associated sequence table header and data fragment headers may be validated.

[0047] Pseudo code of power loss recovery initialization in the non-volatile memory according to an embodiment may be:

```
If (transaction begin but not end) {
   Scan the NV-memory to invalidate all new data
}
Else {
   Go through sequence table(s) to validate all new
data
}
```

[0048] It shall be well understood by a skilled person in the art that the above-described method may be applied to write a part of a file to the non-volatile memory, but not whole of the file. Then, other parts of the file may be written by using other memory writing methods. In such case, the transaction indicator may indicate begin and end states of a transaction for writing a part of a file to the non-volatile memory. Then, a sequence table of the part of the file may be transferred from the volatile memory to the non-volatile memory if the sequence table is full or all of data fragments of that part of the file have been written to the non-volatile memory.

[0049] Fig. 6 illustrates the comparison between two memory writing methods wherein a file to be written comprises 64 data fragments and the sequence tables have a sequence table size limit of 64 entries. The upper part of Fig.5 shows the

steps taken to write the file of 64 data fragments according to a first writing method, while the lower part of Fig. 5 shows those according to a second writing method.

[0050] It can be seen from Fig. 6 that according to the first writing method, in order to write the file of 64 data fragments, the device 100 as shown in Fig. 1 may perform 256 bit twiddling, 128 word programming and 64k bytes buffer programming operations, in which the bit twiddling and word programming operations consumes a relatively large amount of time. However, according to the second memory writing method, in order to write the file with the same size, the device 100 as shown in Fig. 1 may only perform 132 bit twiddling, 0 word programming and 64.5k bytes buffer programming operations, which may significantly save file writing time. It shall be understood that the device 100 may be implemented to utilize both the first file writing method of the second writing method. It shall also be understood that either memory writing method may be applied to write a partial file instead of a whole of the file.

[0051] Although the present invention has been described in conjunction with certain embodiments, it shall be understood that modifications and variations may be resorted to without departing from the spirit and scope of the invention as those skilled in the art readily understand. Such modifications and variations are considered to be within the scope of the invention and the appended claims.